FONDAMENTI DI INFORMATICA

QUARTA LEZIONE

Prof. Alfredo Accattatis

(accattatis@ing.uniroma2.it)

Tutor:

Prof. Vittorio Emanuele Calafiore

(vcalafio@gmail.com)

Accedere alle componenti (indici)

- Indica le modalità di accesso e modifica degli elementi di un vettore
- Gli elementi in un vettore sono numerati sequenzialmente, in MATLAB iniziano dal valore 1 (altri linguaggi da 0; es.: C++)
- Sintassi:
  - **`v(index)`** restituisce l'elemento (o gli elementi) alla(e) posizione(i) specificata(e) dall'*indice*
  - **`v(index)=value`** modifica gli elementi alla(e) posizione(i) specificata(e) dall'indice.
- Il vettore indice può contenere sia valori numerici che logici

```
>> A=1:3:12
A =
     1     4     7    10
>> A(1) %legge primo elemento
ans =
     1
>> B=[2,4]
B =
     2     4
>> A(B)
ans =
     4    10
```

B: vettore indice

Un esempio di vettore indice (index vector)

```
>> A=randn(1,10) %crea A riempita con valori distribuiti normalmente
A =
  Columns 1 through 7
   8.8840e-01  -1.1471e+00  -1.0689e+00  -8.0950e-01  -2.9443e+00
1.4384e+00   3.2519e-01
  Columns 8 through 10
  -7.5493e-01   1.3703e+00  -1.7115e+00
>> A=round(A)
A =
     1    -1    -1    -1    -3     1     0    -1     1    -2
>> iv=2:2:10
iv =
     2     4     6     8    10
>> A(iv)
ans =
    -1    -1     1    -1    -2
>> A(2:2:10)
ans =
    -1    -1     1    -1    -2
```

Indici Numerici

- Il vettore indice può avere qualsiasi lunghezza
- Deve contenere solo numeri interi positivi
- Il valore del vettore indice deve seguire le seguenti regole:
  - In lettura tutti i valori dell'indice devono essere:

    1 <= elemento <= length(vector)
  - In scrittura, tutti i valori dell'indice devono essere:

    1 <= elemento

Esempio

```
>> B = [-1 0 1 2 3];

>> C=[1 2 4 5 1 1];

>> B(C)
ans =
    -1     0     2     3    -1    -1
```

Effettuare «sostituzioni» di valori negli array o aggiunta di nuovi elementi (aumento delle dimensioni)

- Può presentarsi l'esigenza di modificare dei valori
- Questo accade normalmente in molti algoritmi
- MATLAB rende questo compito molto semplice
- Esistono alcune semplici regole

Regole di sostituzione degli elementi di un Array (vettori e matrici)

```
>> A=1:3:12
A =
     1     4     7    10
>> A(7)
??? Index exceeds matrix dimensions.
>> B=[2,4];
>> A(B)=[0,0]
A =
     1     0     7     0
>> A(4)=99
A =
     1     0     7    99
>> A(7)=99
A =
     1     0     7    99     0     0    99
```

Regole di sostituzione degli elementi di un Array (vettori e matrici)

1. Effettuare una sostituzione oltre la fine del vettore implica l'estensione automatica della lunghezza.
2. Tutti gli elementi non esplicitamente sostituiti rimangono invariati.
3. Gli elementi oltre la precedente lunghezza e non direttamente assegnati vengono messi al valore 0 (zero).

```
>> A(7)=99
A =
     1     0     7    99     0     0    99
```

Indici logici

- Il “vettore indice” deve essere di dimensioni minori o uguali lunghezza del vettore originale
- Deve contenere valori logici (true o false)
- L’accesso agli elementi del vettore è effettuato tramite la posizione relativa del vettore logico
  - In lettura, vengono restituiti solo gli elementi corrispondenti all’indice con valore “true”
  - In fase di sostituzione degli elementi, vengono sostituiti solo quelli corrispondenti all’indice con valore “true”
  - Attenzione- I vettori logici sono visualizzati nella command window come 0 e 1 ma non sono assolutamente valori numerici!

```
A =
     1     0     7    99
     0     0    99
```

```
>> a=true
a =
     1
>> a=false
a =
     0
>>
mask=[true,false,true,true]
mask =
     1     0     1     1
>> A(mask)
ans =
     1     7    99
>> A(mask)=[66,66,66];
>> A
A =
    66     0    66    66     0
0    99
```

Il tipo logico è differente dal tipo double!!!

```
>> mask=[0 1 0 1]
mask =
     0     1     0     1

>> whos mask
  Name        Size                Bytes  Class     Attributes

  mask        1x4                    32  double

>> A
A =
     1     4     7    10
>> A(mask)
??? Subscript indices must either be real positive
integers or logicals.
```

“Shortening”, ridurre le dim. di un Array

- **Non è mai effettivamente necessario; è preferibile estrarre i dati voluti tramite indici piuttosto che rimuovere i dati non utili**
- Cambiare le dimensioni di un vettore, ed in generale di una struttura dati, può portare a problemi “logici”
- I problemi possono verificarsi in particolari sezioni dell’algoritmo: consigliabile pertanto usare sempre le funzioni SIZE e LENGTH ed in generale MAI “tenere a mente” le dimensioni ma affidiamoci alle primitive
- Quando è invece utilissimo: per esempio nel caso in cui stiamo manipolando grosse quantità di dati (es. Software che gestiscono riprese video) allora può convenire modificare le dimensioni (risparmio di memoria)

"Shortening", ridurre le dim. di un Array

- Può essere ottenuto applicando l'operatore "vettore vuoto" (usando questo operatore si crea un vettore, appunto, vuoto) ai singoli elementi di un vettore, oppure applicandolo alle intere righe o colonne.
- Attenzione: eliminare un elemento è diverso che metterlo al valore zero!

```
>> vec = 3:5
vec =
    3    4    5
>> vec(2)=[] %rimuove il secondo elemento
vec =
    3    5
```

Esempio…Creiamo un vettore vuoto….

```
>> E=[]
>> whos E
  Name    Size Bytes  Class     Attributes
  E       0x0  0      double
```

Creare un vettore colonna

- E' sufficiente interporre un punto e virgola tra i valori (invece di spazio o virgola), usando la consueta rappresentazione tra parentesi quadre;
-
```
>> c = [2; 5; 7; 1]
c =
2
5
7
1
```
- Non c'è modo diretto per usare l'operatore due punti nel caso di vettori colonna…
- …però si può sempre creare un vettore riga e trasporlo

```
>> r=1:3;  % crea un vettore riga
>> c = r'  % trasposizione
c =
   1
   2
   3
```

Operazioni su vettori

Tre tecniche si estendono automaticamente da quelle disponibili sui valori scalari:

- Operazioni aritmetiche
- Operazioni logiche
- Funzioni di libreria

Due tecniche sono invece specifiche per gli array in generale ed i vettori in particolare:

- Concatenazione
- Indicizzazione generalizzata

Precedenza degli operatori

- Descrizione degli operatori e loro precedenza
- http://it.mathworks.com/help/matlab/matlab_prog/operator-precedence.html?searchHighlight=operator%20precedence&s_tid=doc_srchtitle

Table 5.2: Operator precedence

PRECEDENCE	OPERATOR
1	Parentheses ()
2	Transpose (.'), power (.^), matrix power (^)
3	Unary plus (+), unary minus (−), logical negation (∼)
4	Multiplication (.*), right division (./), left division (.\), matrix multiplication (*), matrix right division (/), matrix left division (\)
5	Addition (+), subtraction (−)
6	Colon operator (:)
7	Less than (<), less than or equal to (≤), greater (>), greater than or equal to (≥), equal to (==), not equal to (∼=)
8	Element-wise AND, (&)
9	Element-wise OR, (|)

Operazioni aritmetiche

Nella Command window, digitare le seguenti righe:

```
>> A = [2 5 7 1 3];
>> A + 5
ans =
7 10 12 6 8
>> A .* 2
ans =
4 10 14 2 6
>> B = -1:1:3
B =
-1 0 1 2 3
```

Operazioni aritmetiche(continued)

```
>> A .* B % moltiplicazione elemento per elemento
ans =
-2  0  7  2  9
>> A * B % moltiplicazione tra matrici!!
??? Error using ==> mtimes
Inner matrix dimensions must agree.
>> A * B'
 ans =
     16
>> C = [1 2 3]
 C =
     1  2  3
>> A .* C % A e C devono avere la stessa lunghezza
??? Error using ==> times
Matrix dimensions must agree.
```

Operazioni logiche

```
>> A = [2 5 7 1 3];
>> B = [0 6 5 3 2];
>> A >= 5
ans =
0 1 1 0 0
>> A >= B
ans =
1 0 1 0 1
>> C = [1 2 3]
>> A > C
??? Error using ==> gt
Matrix dimensions must agree.
```

Operazioni logiche (continued)

```
>> A = [true true false false];
>> B = [true false true false];
>> A & B
ans =
1  0  0  0
>> C = [1  0  0];   % NON è un vettore logico
>> A(C)
??? Subscript indices must either be real positive
 integers or logicals.
```

Ora parleremo di:

- BLOCCHI di CODICE

Blocchi di codice

- Un blocco di codice è una collezione di zero o più istruzioni MATLAB identificate da una o due ragioni:
  1. Le si vuole eseguire solo sotto determinate circostanze, o…
  2. …si vogliono ripetere un certo numero di volte
- Alcuni linguaggi identificano i blocchi di codice racchiudendoli dentro parentesi graffe , altri dal livello di indentazione del testo
- MATLAB usa l'occorrenza di parole chiave nel testo per definire implicitamente i limiti del blocco stesso:
  - **`if, switch, while, for, case, otherwise, else, elseif, end`**
- **Le parole chiave** sono identificate, nel text editor di MATLAB, in blu. Non sono parte del blocco di codice ma servono in qualità di:
  - istruzioni che definiscono cosa fare con il blocco di codice
  - delimitatori che definiscono i confini del blocco di codice

Blocco di codice: esempio

- Un blocco di codice è una collezione di zero o più istruzioni MATLAB identificate da uno o due motivi :
  1. Le si vuole eseguire solo sotto determinate circostanze, o…
  2. …si vogliono ripetere un certo numero di volte

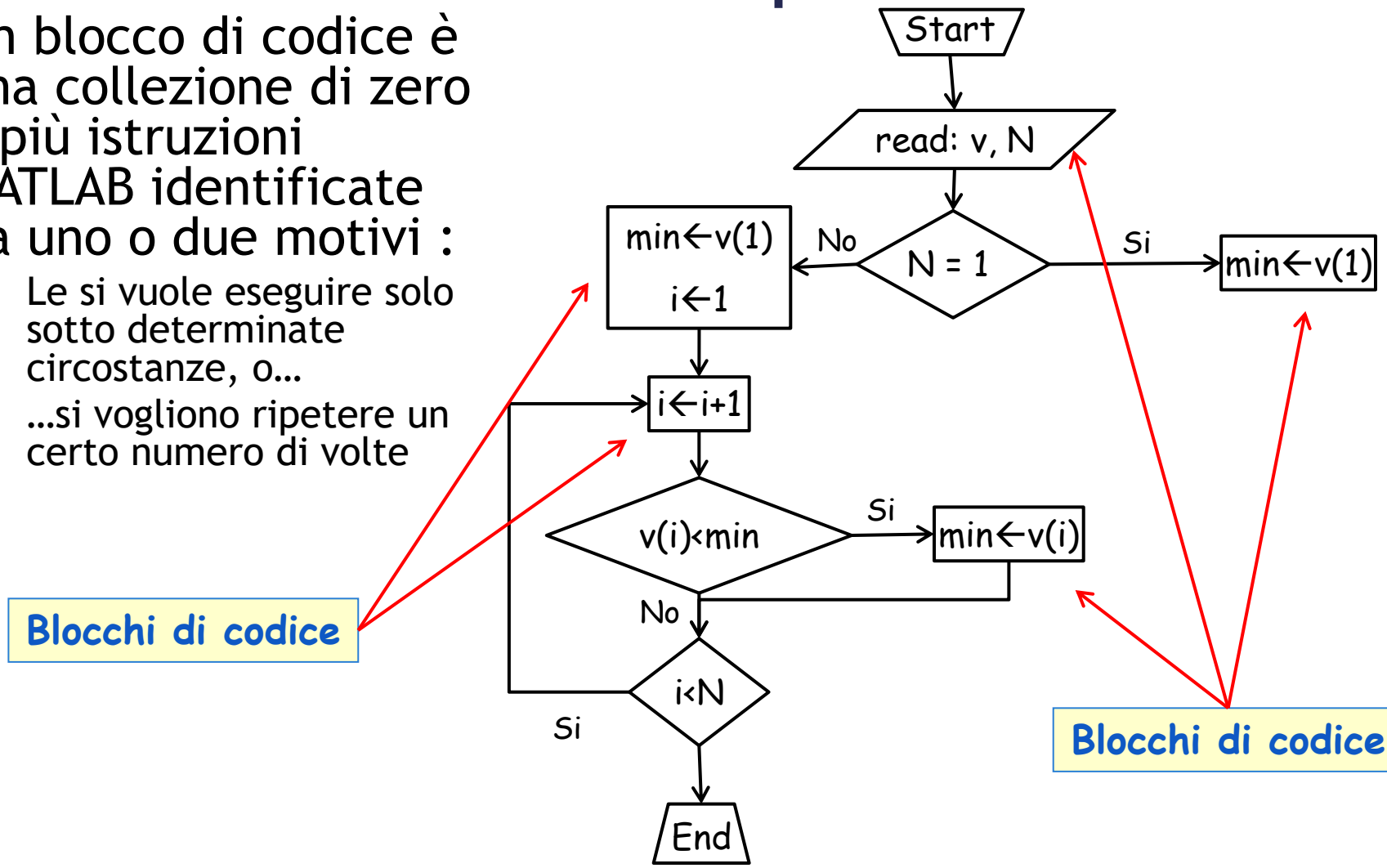
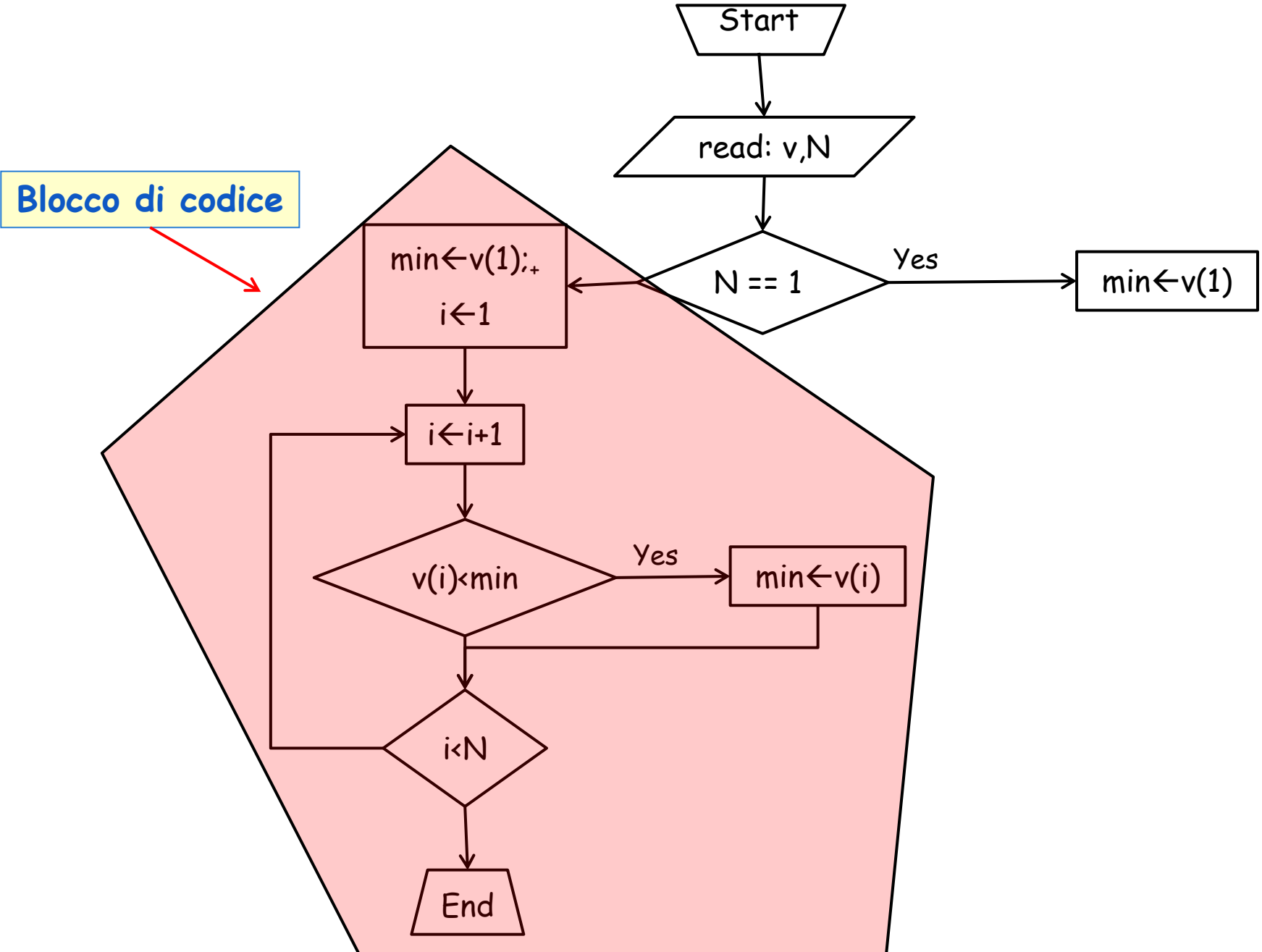
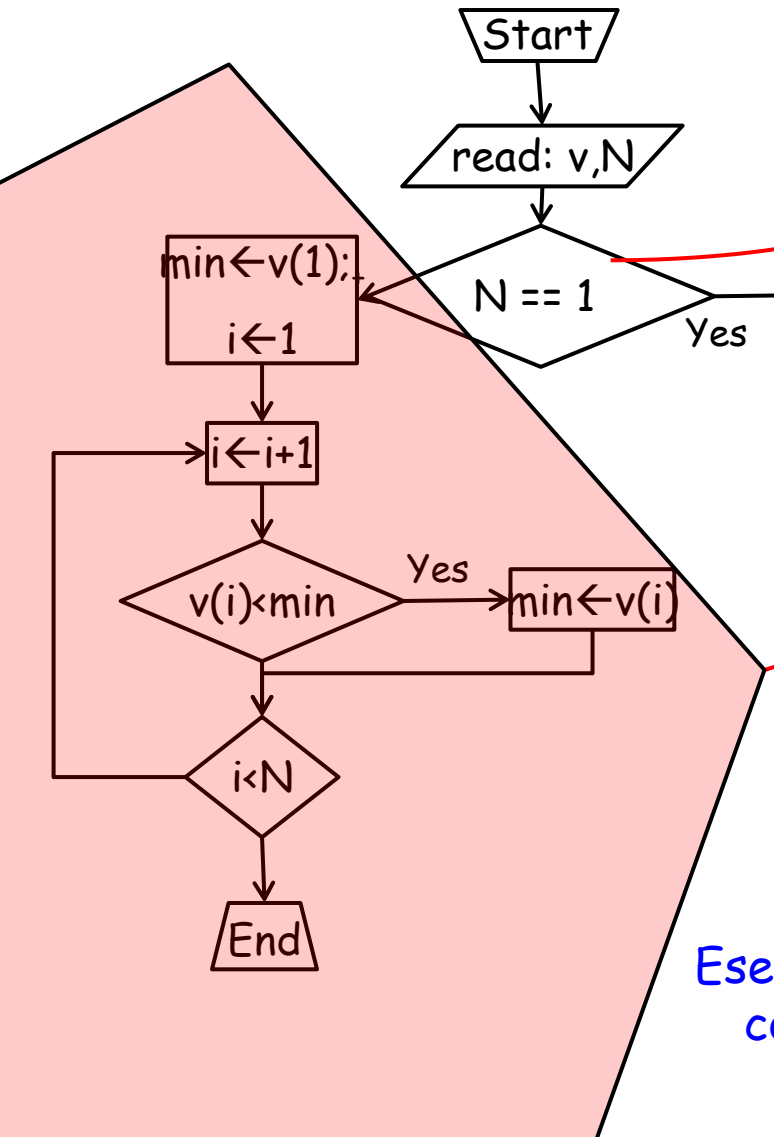


Diagramma di flusso per trovare il valore minimo di un vettore


Blocco di codice
Start
read: v,N
N == 1
Yes
min←v(1)
min←v(1);
i←1
i←i+1
v(i)<min
Yes
min←v(i)
i<N
End

Blocco di codice: esempio

min(v)

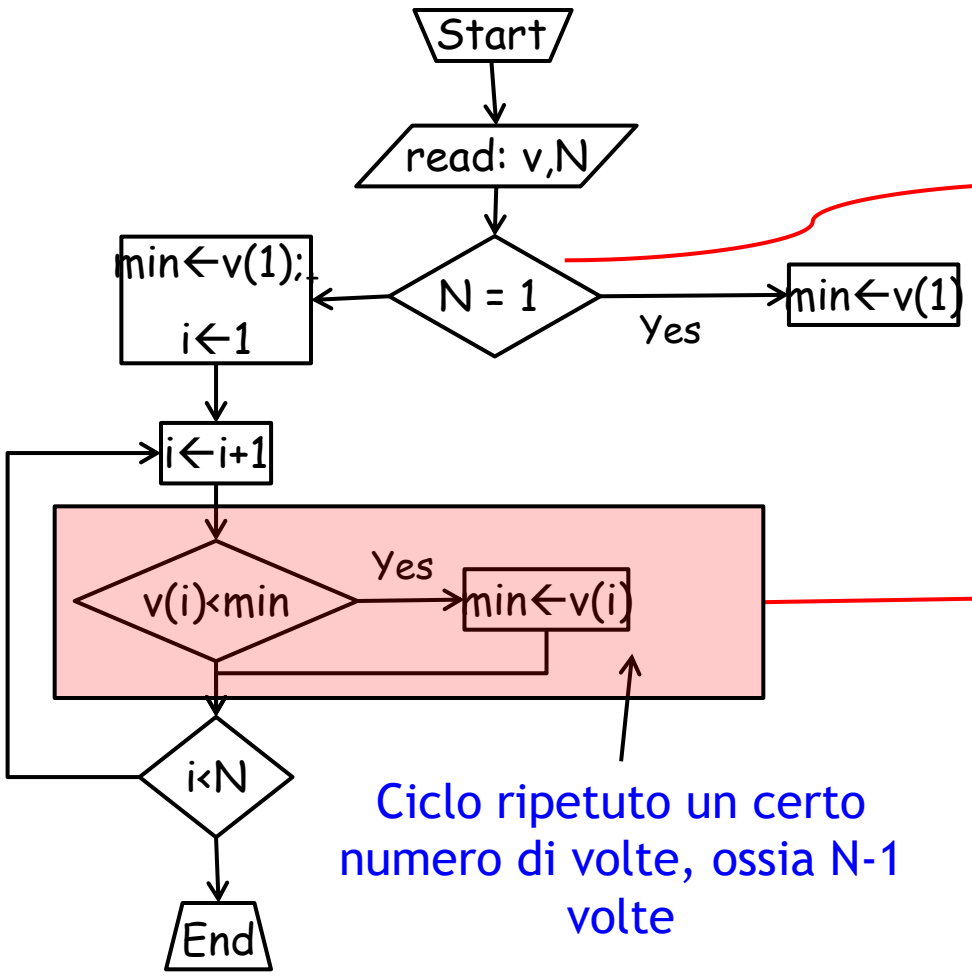



```
v=[4 5 2 19 5 7 8];
N=length(v);
if N==1
     mymin=v(1);
else
     mymin=v(1);
     i=2;
     while (i<=N)
          if v(i)<mymin
               mymin=v(i);
          end
          i=i+1;
     end
end
```

Eseguito solo sotto certe condizioni:ossia N > 1

Blocco di codice: esempio

min(v)




```
v=[4 5 2 19 5 7 8];
N=length(v);
if N==1
     mymin=v(1);
else
     mymin=v(1);
     i=2;
     while (i<=N)
          if v(i)<mymin
               mymin=v(i);
          end
          i=i+1;
     end
end
```

Ciclo ripetuto un certo numero di volte, ossia N-1 volte

Blocchi di codice: identificazione

- Alcuni linguaggi identificano i blocchi di codice racchiudendoli tra parentesi quadre, i.e., {*blocco*}; altri tramire il livello di "indentazione" del testo

C

```
for(i=1;i<=m;i++)
{
 for(j=1;j<=n;j++)
 x[i][j]=0;
}
```

Python

```
x = 1
while x < 5:
    print ('Valore n. ')
    print (x)
    x = x + 1
print ('finito')
```

Blocchi di codice: identificazione

- MATLAB usa l'occorrenza di parole chiave nel testo per definire implicitamente I limiti del blocco stesso :
  - **if, switch, while, for, case, otherwise, else, elseif, end**

MATLAB

```
for i=1:length(log3s_300);
    for iterK=1:length(K)
      start=1+(i-1)*(tau1)*W;
      lambda=log3s_300(start:start+W*tau1-1);
      Xmax=Tmax*max(lambda)/(Tmax*mu-1)*2;
      filename=sprintf('output/%0.0fhK%sWl%0.0f',tau1, strK{iterK},start);
      [x,TC,neval,XFlg, cost, avgViol] = optimalAllocation();
    end
end
```

Ora ci occupiamo di...

- Istruzioni condizionali (**`if, if-else-elseif, switch`**)
- Funzioni
- Cenni programmazione procedurale, indentazione
- Cicli / Iterazioni (**loop**): permettono di ripetere una sequenza di istruzioni un determinato numero di volte oppure finchè non si verifica una determinata condizione:
  - **`for`**
  - **`for`** nidificati
  - **`while`**

Istruzioni condizionali

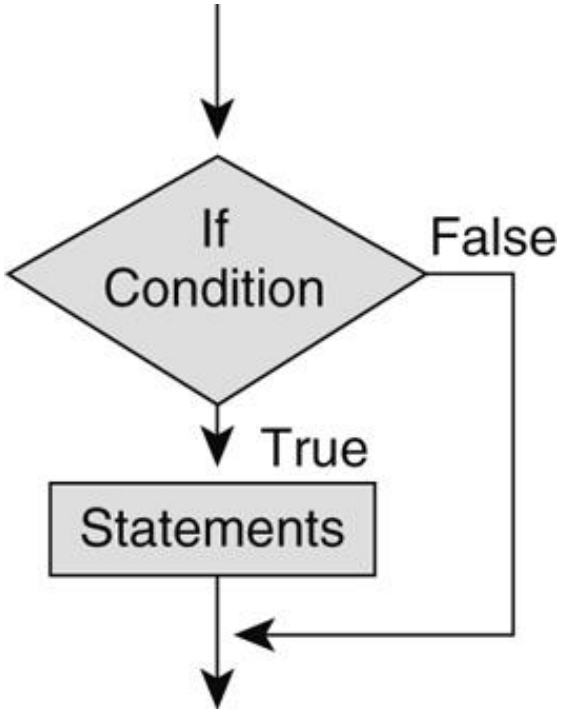
- Le istruzioni condizionali permettono di fare delle scelte, ossia determinare se blocchi di codice sono eseguiti o no (ossia come scegliere tra due o più cammini)

- Matlab mette a disposizione:
  - **`if`**
  - **`if`** con **`else`**
  - **`if`** con **`elseif`**
  - **`switch`**

Istruzioni condizionali

- Sono richiesti due elementi base:
  - Un'espressione logica (condizione)
  - Un blocco di codice
- Se l'espressione risulta vera (true) il blocco di codice verrà eseguito
- Altrimenti l'esecuzione continua saltando il blocco, ossia alla prima istruzione successiva al blocco di codice
- La forma generale dell'**if** è

```
if condition
     <azione>
end
```



Esempio:

```
if v(i)<mymin
       mymin=v(i);
end
```

Esempio di istruzione IF

- Uno script che invita l'utente a digitare un numero e successivamente calcola e stampa la radice quadrata. Se l'utente digita un numero negativo, verrà usato il valore assoluto

```
num = input('Digitare un numero: ');
if num < 0
        disp('Verrà usato il valore assoluto. ');
        num = abs(num);
end
fprintf('La radice quadrata di %.1f is %1f\n', num, sqrt(num));
```

Una "clausola condizionale" più complessa

- Consideriamo il problema di determinare I migliori dieci giocatori:
- Definiamo
  - yearsInLeague: vettore contenente gli anni passati a giocare in campionato
  - errorPerYears: vettore contenente gli errori per anno
  - plateApparence: vettore contenente in numero di partecipazioni per anno

```
if yearsInLeague(i)>=5 && errorPerYears(i)<=10 && plateApparence(i)>100
   selectThePlayer(i)
end
```

Operatore LOGICO and

A && B == true se A=true and B=true

A && B == false altrimenti

Operatore LOGICO and doppio = &&

Vale per gli SCALARI ed e “short circuit” ossia valuta solo la

prima condizione che determina il risultato (stesso risultato con risparmio di risorse)

Infatti AND è vero solo se TUTTI gli operandi sono veri, al primo falso smette di valutare

Operatore LOGICO and singolo &

Vale per i vettori e matrici, valutando “and” componente per componente ;

Se usato per gli scalari NON è short circuit

```
a = 1     2     3     4     5

b = 6     7     8     9    10

>> a & b

ans =

  1×5 logical array

   1   1   1   1   1

>> a && b
```

Operands to the logical AND (&&) and OR (||) operators must be convertible to logical scalar values

Istruzione if-else

- Come scegliere tra due opzioni?
- La forma generale dell'istruzione **if-else** è

```
if <condizione>
    <azione1>
else
    <azione2>
end
```

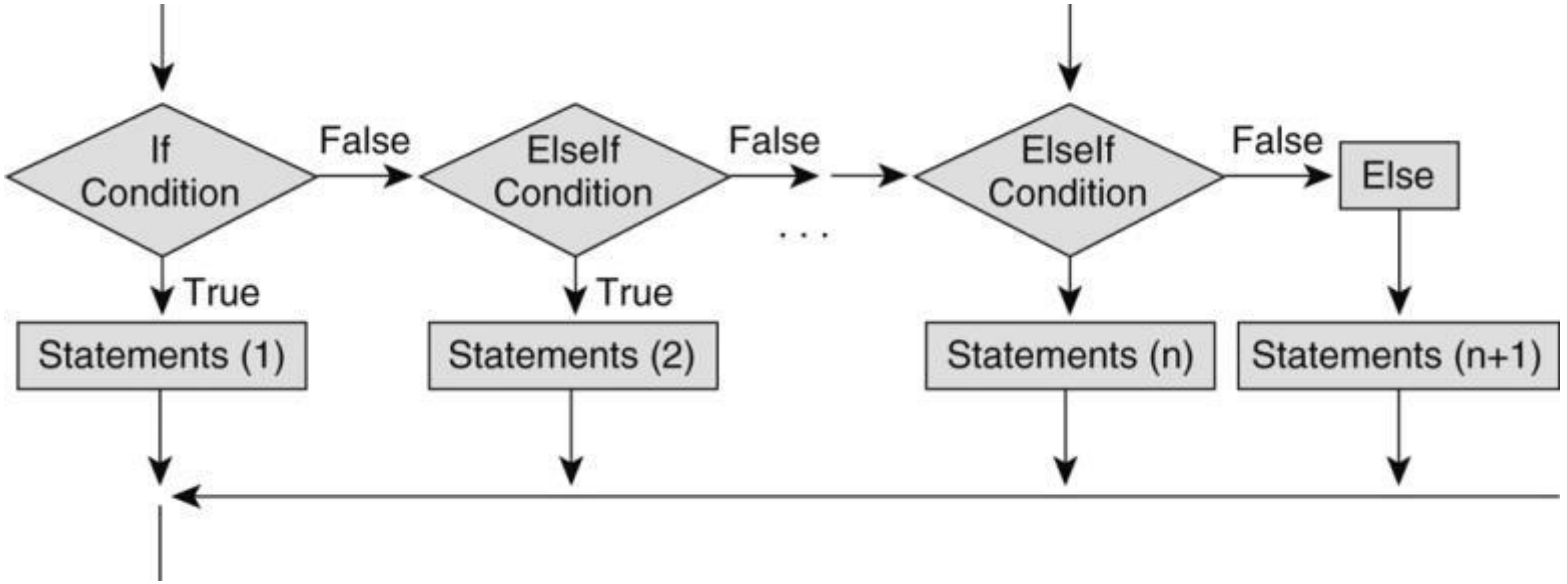
Esempio: stampare o no un numero casuale (range 0..1) se minore di 0.5

```
r = rand();
if r < 0.5
    disp('Numero < 0.5');
else
    disp('Numero >= 0.5');
end
```

Esempio di istruzione `if-else`

```
v=[4 5 2 19 5 7 8];
N=length(v);
if N==1
    mymin=v(1);
else
    mymin=v(1);
    i=2;
    while (i<=N)
        if v(i)<mymin
            mymin=v(i);
        end
        i=i+1;
    end
end
```

IF “composti”



- Per mezzo del costrutto `elseif` , abbiamo la possibilità di scegliere tra opzioni multiple
- Lo stesso può essere ottenuto usando “if” separati o “if-else” nidificati, ma l’uso di elseif rende tutto più semplice

Istruzione if / elseif

- La forma generale per l'istruzione **if** con **elseif** é:

```
if <espressione logica 1>
      <blocco di codice 1>
elseif <espressione logica 2>
      <blocco di codice 2>
.
.
.
elseif <logical expression n>
      <blocco di codice n>
else
      <blocco di codice di default>
end
```

Esempi con elseif

```
if temperature > 100
      disp('Too hot - equipment malfunctioning. ')
elseif temperature > 90
      disp('Normal operating range. ')
elseif temperature > 50
      disp('Below desired operating range. ')
else
      disp('Too cold - turn off equipment. ')
end
```

```
if x < -1
   y = 1;
elseif  x <= 2
   y = x^2;
else
   y = 4;
end
```

Scegliere tra opzioni multiple

- Scegliere tra opzioni multiple può essere ottenuto usando if separati oppure if-else nidificati, ma usare elseif è la maniera più facile ed efficiente

Uso di `elseif`

```
if x < -1
        y = 1;
elseif x <= 2
        y = x^2;
else
        y = 4;
end
```

Uso di `if` distinti

```
if x < -1
        y = 1;
end
if x >= -1 && x <= 2
        y = x^2;
end
if x > 2
        y = 4;
end
```

Uso di `if-else` nidificati

```
if x < -1
        y = 1;
else
        if x <= 2
                y = x^2;
        else
                y = 4;
        end
end
```

Esempi ulteriori

```
% esempio di uso di if
 day = input('Scelta giorno(1-7): ');
 if day == 7          % Sabato
    stato = 'feriale'
 elseif day == 1      % Domenica
    stato = 'feriale'
 else
    stato = 'lavorativo'
 end
```

```
% altro esempio con if
 grade = input('Quanti gradi? ');
 if grade >= 90
     letter = 'A'
 elseif grade >= 80
     letter = 'B'
 elseif grade >= 70
     letter = 'C'
 elseif grade >= 60
     letter = 'D'
 else
     letter = 'F'
 end
```

- Un blocco di codice può essere memorizzato come uno script MATLAB e successivamente eseguito

Ancora... if...elseif...else...end

```
% Visualizza testo indicando se x è:
% scalare, vettore, matrice
      [m,n] = size(x);
      if m==n && m==1
            disp('scalare')
      elseif m==1 || n==1
            disp('vettore')
      else
            disp('matrice')
      end
```

Perché funziona?

operatore logico OR

A || B==true se almeno A=true o B=true

A || B==false altrimenti

Osservazioni generali

- Una espressione logica è una qualsiasi espressione (dichiarazione) che restituisce un risultato logico.
- Se l'argomento (e risultato) è un vettore logico **v**, vengono valutati tutti gli elementi, ossia agisce se sono tutti allo stesso valore logico

```
%Istruzione if con vettori logici
 A = [true true false]
 if A
    % non eseguito
 end
 A(3) = true;
 if A
    % eseguito
 end
```

- L'indentazione, in generale, NON ha effetto sull'esecuzione del programma, piuttosto aiuta a rendere più chiaro il flusso logico
- L'editor di MATLAB cerca di creare indentazioni automatiche

Indentare!

```
Max = A(1);
Index = 1;
for ind = 1:length(A)
      if A(ind) > Max
          Max = A(ind);
          Index = ind;
      end
end
```

```
Max = A(1);
Index = 1;
for ind = 1:length(A)
if A(ind) > Max
Max = A(ind);
Index = ind;
end
end
```

Istruzione switch

- La forma generale è:

```
switch <expression>
  case <case specification 1>
    <code block 1>
  case <case specification 2>
    <code block 2>
          .
          .
  case <case specification n>
    <code block n>
  otherwise
    <default code block>
end
```

Considerazioni generali

- L'istruzione switch controlla che l'espressione abbia un riscontro esatto con uno dei "casi"
- Un singolo caso può contenere valori multipli tra parentesi graffe {…}
- Il "caso di default" comprende tutti i casi non compresi nei casi specifici
- Il caso di default dovrebbe, di norma, comprendere i casi anomali

```
%Esempio di switch

month = input('Digitare un mese (1-12): '
 switch month
    case {9, 4, 6, 11}
        % Sept, Apr, June, Nov
        days = 30;
    case 2                % Feb
          if leapYear %vero se anno bises
           days = 29;
          else
           days = 28;
          end
    case {1, 3, 5, 7, 8, 10, 12}
        % altri mesi
        days = 31;
    otherwise
          disp('Indice mese errato')
          days=0;
 end
 days
```

if-else versus switch

- Problema
  - Se la variabile “interval” è minore di 1, setta il valore di “xinc” a interval/10; altrimenti settala al valore 0.1

IF…ELSE

```
if interval < 1
      xinc = interval/10;
else
      xinc = 0.1;
end
```

SWITCH

```
switch interval < 1
      case 1
            xinc = interval/10;
      case 0
            xinc = 0.1;
end
```

if-else versus switch: hint

IF…ELSE

```
if interval < 1
        xinc = interval/10;
else
        xinc = 0.1;
end
```

=

IF…ELSE

```
a=interval < 1;
if a
        xinc = interval/10;
else
        xinc = 0.1;
end
```

SWITCH

```
switch interval < 1
    case 1
        xinc = interval/10;
    case 0
        xinc = 0.1;
end
```

=

SWITCH

```
a=interval < 1;
switch a
    case 1
        xinc = interval/10;
    case 0
        xinc = 0.1;
end
```

Funzioni User-defined

- Abbiamo già incontrato (ed usato) molte funzioni "built-in" di matlab: E.g., `linspace`, `size`, `length`, `disp`, `plot;`
- E' possibile definire funzioni per proprio uso e consumo;
- Le funzioni definite dall'utente possono essere "chiamate" dalla Command Window oppure all'interno di uno Script.
  - Per chiamare una function si usa il nome, seguito dalla lista degli argomenti di ingresso (**input argument(s)**) tra parentesi
  - E.g., `length(vec)`
  - La funzione restituisce gli "argomenti di uscita" (**output argument(s)**)
  - E.g., `lv = length(vec)`
- Per il momento consideriamo funzioni definite dall'utente che restituiscono un singolo valore

Definizione di funzione

- Per ora memorizziamo ogni funzione in file .M separato (come abbiamo già fatto per gli script)
- Una function consiste di:
  - Un **header** (la prima linea) che comprende:
    - La parola chiave "`function`"
    - Il nome degli parametri di uscita (argomenti), seguito dall'operatore di assegnazione (=); la funzione sta "ritornando" un valore
    - Il **nome della funzione** (dovrebbe essere lo stesso del nome del file .M che contiene la definizione della funzione)
    - Gli argomenti (parametri) di ingresso tra parentesi
  - Un **commento** che descrive cosa fa la funzione
  - Il corpo della funzione, che include tutte le istruzioni ed eventualmente valorizza i parametri di uscita (end)

Esempio di funzione

- Vogliamo definire una funzione che crea un vettore di interi di valori crescenti e compresi tra mymin e mymax
- La nostra funzione avrà due parametri di ingresso (mymin e mymax) e restituisce un vettore con la lista di valori compresi tra mymin e mymax
- In primo luogo, dovremo assicurarci che mymin sia minore di mymax
- Se la condizione precedente non è verificata, dovremo cambiare i valori prima di creare il vettore

Esempio di funzione(2)

Parola riservata **function**

Argomenti d'uscita

Due argom. ingresso

Nome della funzione

Assegnazione

```
function outvec = createvec(mymin,mymax)
% createvec crea un vettore di valori che variano tra un
% minimo  ed un massimo
% restituisce il vettore

% se il minimo non è minore del massimo, scambia i valori
if mymin > mymax
        temp = mymin;
        mymin = mymax;
        mymax = temp;
end

outvec = mymin:mymax;
end
```

- Salviamo la funzione in un M-file (`createvec.m`, stesso nome della funzione)
- Chiamiamo la funzione

```
>> createvec(7,3)

ans =

    3  4  5  6  7
```

Esempio di funzione(3)

- Notare che per scambiare tra loro i valori, una terza variabile (chiamata variabile temporanea) è richiesta (altrimenti il valore si perde!)
- **Esempio: mymin = 5 e mymax = 3;**
- Attenzione: il codice seguente è sbagliato! Errore tipico!

```
mymin = mymax;          % mymin = 3
mymax = mymin;          % mymax = 3
```

- Ecco il codice corretto

```
temp = mymin;           % temp = 5
mymin = mymax;          % mymin = 3
mymax = temp;           % mymax = 5
```

Esempio di funzione(4)

- Parametri formali : le variabili dichiarate nella definizione di funzione

- Parametri attuali : i valori che passiamo nei parametri di ingresso e le variabili «esterne» che usiamo per accogliere i valori restituiti dalle function :

- **`function outvec = createvec(mymin,mymax)`**
- Outvec, mymin e mymax parametri formali
- >> uscita = createvec(1,10)
- Uscita, 1 e 10 parametri attuali

Come creare una funzione: indentare!

```
function [ Max, Index ] = myMax( A )
      Max = A(1);
      Index = 1;
      for ind = 1:length(A)
            if A(ind) > Max
                Max = A(ind);
                Index = ind;
            end
      end
```

Manca qualcosa?

I commenti!

Senza/con indentazione

```
function [ Max, Index ] = myMax( A )
Max = A(1);
Index = 1;
for ind = 1:length(A)
if A(ind) > Max
Max = A(ind);
Index = ind;
end
end
```

```
function [ Max, Index ] = myMax( A )
    Max = A(1);
    Index = 1;
    for ind = 1:length(A)
        if A(ind) > Max
            Max = A(ind);
            Index = ind;
        end
    end
```

Commenti, che scocciatura

- Nella definizione di funzione, le righe dopo la prima, dovrebbero essere commenti, che spiegano cosa fa la funzione che si sta definendo.

Provate a digitare

>> help createvec

Provate a digitare

>> which <nome di una funzione qualsiasi nota>

Concetto di ITERAZIONE

- **L'Iterazione** permette la ripetizione controllata di un blocco di codice ("loop")
- Istruzioni di controllo all'inizio del blocco di codice definisce i limiti (ed il modo) della ripetizione :
  - L'istruzione di loop "**for**" è definita per ripetere un blocco di codice un numero prefissato di volte
  - L'istruzione di loop "**while**" è più flessibile : consente di ripetere il blocco di codice un numero di volte variabile (dipende dalla "condizione")

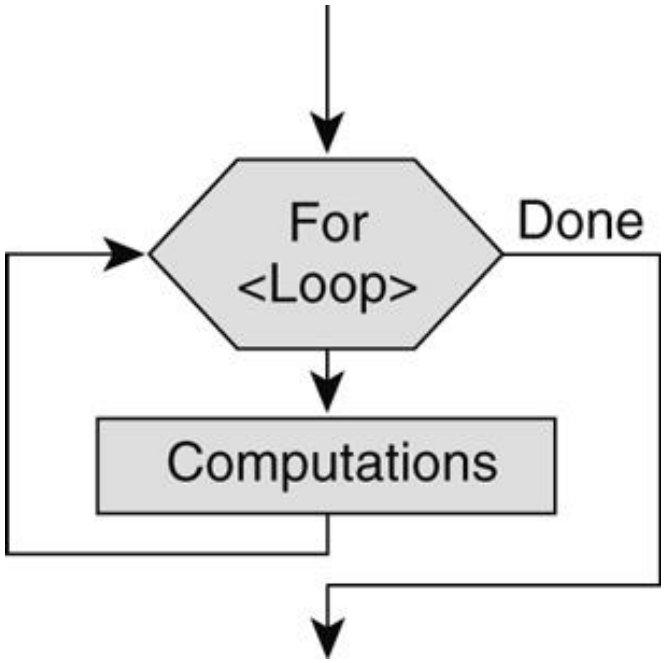
`for` loop

Lo schema base del for loop è:

```
for loopvar = vector
    <blocco di codice>
end
```

Più in generale:

```
for variable = expr
    <blocco di codice>
end
```



Il “for loop” automaticamente setta il valore della variabile di controllo del loop ad ogni giro ed esegue il blocco di codice con quel valore

Esempi di loop

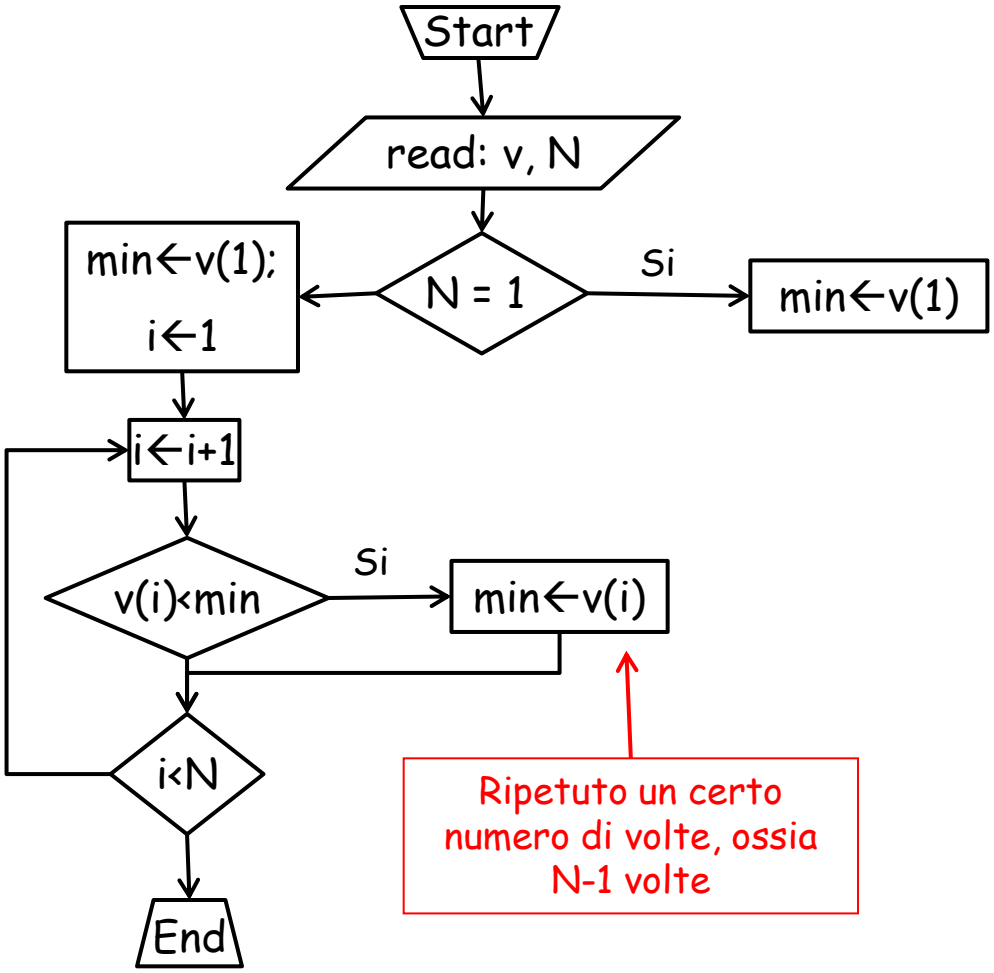
```
% implementa la funzione zeros(1,N)
N=10;
for i=1:N          % specifichiamo il range dei valori
    v(i)=0;        % usando l'operatore colonna
end

% setta gli elementi dispari di  v al valore  1
for i=1:2:N
    v(i)=1;
end
%lo stesso codice di cui sopra può essere scritto come:
for i=[1,3,5,7,9]
    v(i)=1;
end
```

Esempio: ingresso in un `for` loop

- Non è sempre necessario usare il valore della variabile di loop nel blocco di codice!

```
% Questo script ripete l'azione di "prompting" verso
% l'utente per un certo numero di volte, e stampa la
% conferma
for iv = 1:3
    inputnum = input('Digitare un numero: ');
    fprintf('Hai digitato %.1f\n', inputnum)
end
```

min(v), (Trova il minimo di un vettore)




```
v=[4 5 2 19 5 7 8];
N = length(v);
min=v(1);

for i=2:N
    if v(i)<min
        min=v(i);
    end
end
```

Trova il massimo in un vettore

```
%Esempio di for
A = floor(rand(1,10)*100) % vettore iniziale
Max = A(1);
Index = 1;
for ind = 1:length(A)
    if A(ind) > Max
        Max = A(ind);
        Index = ind;
    end
end
fprintf('il massimo valore in A è %d alla posizione
%d\n', Max, Index);
```

Trova il massimo in un vettore - V2

```
%Esempio di for loop
 A = [6 12 6 91 13 6] % vettore iniziale
 Max = A(1);          % primo valore
 for x = A            % itera sul vettore A
     if x > Max       % testa ogni elemento
         Max = x;
     end
 end
 fprintf('max(A) è %d\n', Max);
```

Loop nidificati (innestati)

- Le istruzioni all'interno del loop (azione del loop) può essere una combinazione di qualsiasi istruzione
- Quando sono esse stesse istruzioni di loop, è chiamato loop innestato o ciclo nidificato
- La forma generale di un ciclo nidificato è:

```
for loopvar1 = range1
  %action1 - include il ciclo più interno
  for loopvar2 = range2
    %action2
  end
end
```

Esempio di loop nidificati con for

```
% stampa una rettangolo di stelle
% le dimensioni sono specificate da due variabili
% (numero di righe e colonne)
nrows = input('Digitare numero di righe: ');
ncols = input('Digitare numero di colonne: ');
% cicla sulle righe
for i=1:nrows
  % per ogni riga stampa "ncols" asterischi e va a capo (\n)
  for j=1:ncols
    fprintf('*');
  end
  fprintf('\n');
end
```

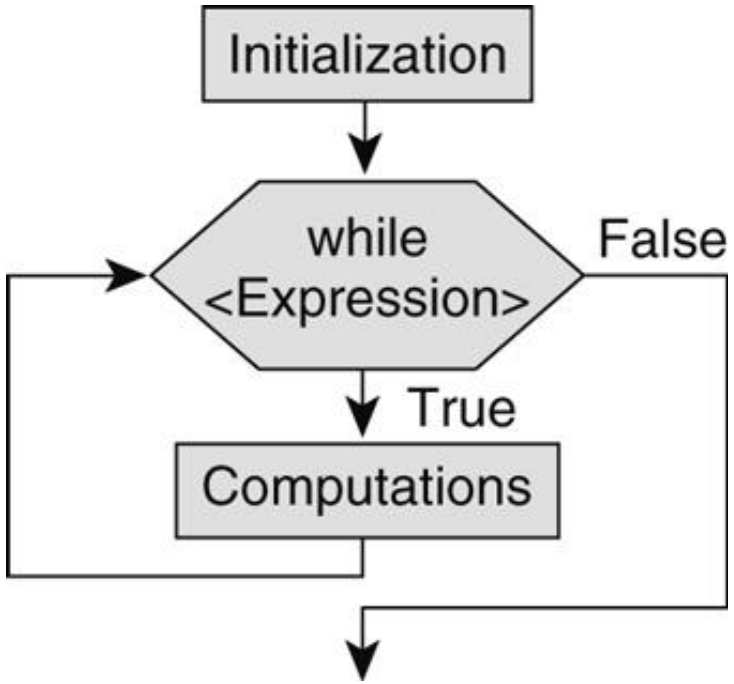
Creiamo la funzione…

Esempio di loop nidificati con for (function)

```
function printBox(nrows, ncols)
% stampa una rettangolo di stelle
% le dimensioni sono specificate da due variabili
% (numero di righe e colonne)

% cicla sulle righe
for i=1:nrows
  % per ogni riga stampa "ncols" asterischi e va a capo (\n)
  for j=1:ncols
    fprintf('*');
  end
  fprintf('\n');
end
```

`while` loop

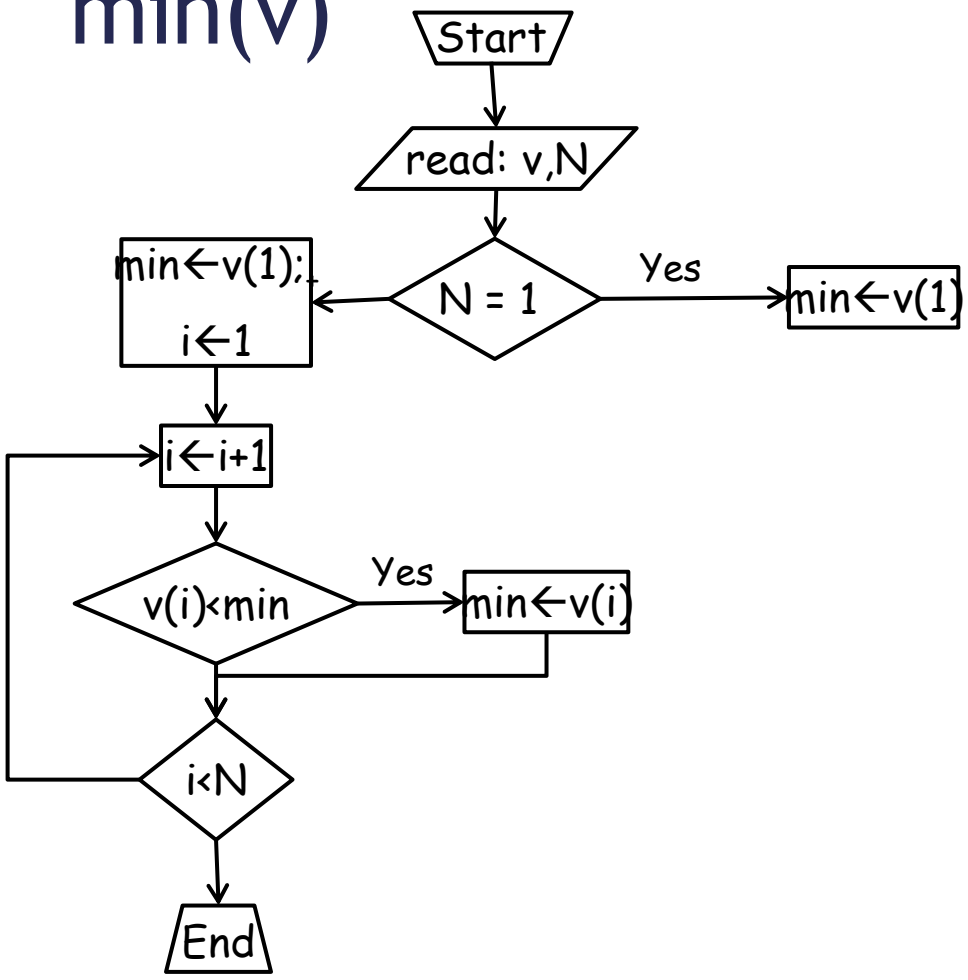
Il blocco di codice è ripetuto fino a che l'espressione logica rimane vera



In generale:

```
<initialization>
while <logical expression>
    <code block>
    % per far terminare il ciclo
    % bisogna modificare l'espressione logica
end
```

min(v)




```
v=[4 5 2 19 5 7 8];
N=length(v);
if N==1
     mymin=v(1);
else
     mymin=v(1);
     i=2;
     while (i<=N)
          if v(i)<mymin
               mymin=v(i);
          end
          i=i+1;
     end
end
```

Homework: creare una funzione che prende in input il vettore **v** e restituisce come output il minimo valore contenuto in **v**

Un esempio ulteriore

```
% Script che calcola ax^2 + bx + c
disp('calcolo ax^2+bx+c')
disp('parametri d'ingresso a, b, c; and x')
a=1; b=1; c=1; x=0;
while a~=0 || b~=0 || c~=0 || x~=0
    disp('Porre a=b=c=x=0 per terminare')
    a = input('Digitare a: ');
    b = input('Digitare b: ');
    c = input('Digitare c: ');
    x = input('Digitare x: ');
    if a==0 && b==0 && c==0 && x==0
        break
    end
    quadratic = a*x^2 + b*x + c;
    disp('Risultato:')
    disp(quadratic)
end
```